SMT-based Scheduling for Multiprocessor Real-Time Systems

Zhuo Cheng*, Haitao Zhang†, Yasuo Tan*, and Yuto Lim*
*School of Information Science, Japan Advanced Institute of Science and Technology, Japan
{chengzhuo, ytan, ylim}@jaist.ac.jp
†School of Information Science and Engineering, Lanzhou University, China
htzhang@lzu.edu.cn

***Abstract*—Real-time system is playing an important role in our society. For such a system, sensitivity to timing is the central feature of system behaviors, which means tasks in the system are required to be completed before their deadlines. Currently, almost all the practical real-time systems are equipped within multiple processors, for which the schedule synthesis to make sure that all the tasks can be completed before their deadlines is known to be an NP complete problem. In this paper, to solve the scheduling problem, we propose a scheduling method based on *satisfiability modulo theories (SMT)*. In the method, the problem of scheduling is treated as a satisfiability problem. The key work is to formalize the satisfiability problem using first-order language. After the formalization, a SMT solver (e.g., Z3, Yices) is employed to solve such a satisfiability problem. An optimal schedule can be generated based on a solution model returned by the SMT solver. Moreover, in the SMT-based scheduling method, we define the scheduling constraints as system constraints and target constraints. Such design makes the proposed method apply more widely compared with existing methods.**

***Keywords*—real-time scheduling, SMT, multiprocessor**

I. Introduction

Real-time system is playing an important role in our society. For example, chemical and nuclear plant control, space missions, flight control, autonomous driving systems, telecommunications, and multimedia systems are all real-time systems [1]. Sensitivity to timing is the central feature of system behaviors, which means tasks in the system are required to be completed before their deadlines. Currently, almost all the practical real-time systems are equipped within multiple processors, for which the schedule synthesis to make sure that all the tasks can be completed before their deadlines is known to be an NP complete problem [4].

Many researches have been conducted to challenge this problem. A comprehensive survey of scheduling for multiprocessor real-time systems can be found in [5]. In [8], the *Proportionate Fair* (Pfair) algorithm was introduced. Pfair is a schedule generation algorithm which is applicable to sets of periodic tasks with implicit deadlines. It is based on the idea of fluid scheduling, where each task makes progress proportionate to its utilization. Pfair scheduling divides the timeline into equal length quanta or slots. Experimental results in [8] showed that the Pfair algorithm is optimal for periodic task sets with implicit deadlines. In [9], authors extended the PFair approach to sporadic task sets, showing that the EPDF (earliest pseudo-deadline first) algorithm, a variant of Pfair, is optimal for sporadic task sets with implicit deadlines executing on two processors, but is not optimal for more than two processors.

Some approaches focus on studying task and messages schedule co-synthesis in switched time-triggered networks. In [10], authors studied time-triggered distributed systems where periodic application tasks are mapped onto different end stations (processing units) communicating over a switched Ethernet network. They try to solve the scheduling problem using a MIP multi-objective optimization formulation. In [11], authors studied the system consisting of communicating event- and time-triggered tasks running on distributed nodes. These tasks are scheduled in conjunction with the associated bus messages by using dynamic and static scheduling methods, respectively. Hitherto, most of the presented methods are either limited to specific task model (e.g., [8, 10] limited to periodic task sets) or simple system architecture (e.g., [9] limited to two processors, [11] simple bus network topologies).

In this paper, we try to solve the scheduling problem for multiprocessor real-time systems in a more generous way, which means the proposed method applies more widely compared with the existing methods. The main contributions of this paper are as follows.

i) We propose a scheduling method based on satisfiability modulo theories (SMT). In this method, the problem of scheduling overload is treated as a satisfiability problem. The key work is to formalize the satisfiability problem using first-order language. We use a *sat model* to represent the formalized problem. This sat model is a set of first-order logic formulas (within linear arithmetic in the formulas) which express all the scheduling constraints that a desired optimum schedule should satisfy. After the sat model is constructed, a SMT solver (e.g., Z3 [6], Yices [7]) is employed to solve the formalized problem. An optimal schedule can be generated based on a *solution model* returned by the SMT solver. The correctness of this method and the optimality of the generated schedule are straightforward.

ii) The proposed SMT-based scheduling method applies more widely compared with existing methods. In the SMT-based scheduling method, we define the scheduling constraints as system constraints and target constraints. It means if we want to design scheduling to achieve other objectives (e.g., maximizing effective processor utilization), only the target constraint needs to be modified. Or, if we want to achieve the same scheduling objective for another real-time system with different system architecture (e.g., different network topolo-

Haitao Zhang is the corresponding author.

978-1-5090-0806-3/16/$31.00 copyright 2016 IEEE
ICIS 2016, June 26-29, 2016, Okayama, Japan

gies), only the system constraints need to be modified. This means the proposed method is flexible and sufficiently general.

The remainder of this paper is organized as follows. In section II, we present the system model and give an example to show the problem of scheduling for multiprocessor real-time systems. The simple introduction of SMT and the details of the SMT-based scheduling are described in section III. Simulation and performance evaluation are shown in section IV. Concluding remarks are given in section V.

II. System Model

In this section, we present system model studied in this paper. For convenience, symbols used throughout the paper are summarized in Table I.

A. Function Set

Function set defines the functions that can be achieved by a real-time system. Let $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$ denote the function set. Each function $F_i \in \mathcal{F}$ is achieved by a series of tasks, represent as *poset* $(T_i, \prec)$, $T_i \neq \emptyset$ denotes the set of the corresponding task, and $\prec$ denotes the dependency relation of tasks in T_i (the detail of the poset will be explained in the next subsection). For real-time systems, when functions are triggered at system time instant r, they are required to be completed before a specific time instant, which is called *deadline*, represented by d_i. Based on above analysis, we define the function $F_i = ((T_i, \prec), r_i, d_i)$.

Note that, unlike many researches on real-time scheduling that set deadlines to tasks, we set deadline to the function level rather than task level. This setting can better reflect the reality that the deadline requirement is for functions of real-time systems, while a function is achieved by a series of tasks cooperated together.

B. Task Poset

A multiprocessor real-time system comprises a set of tasks, denoted by $\mathcal{T}$. For each function, it is achieved by a series of tasks cooperated together. Poset $(T_i, \prec)$ is used to denote such a series of tasks, where $T_i \subseteq \mathcal{T}$ is the task set corresponding to F_i, and $T_i = \{\tau_1, \tau_2, \ldots, \tau_m\}$, where $\tau_j \in T_i$ is a task, and m is the number of tasks. We use $\tau_{i,j}$ to indicate task $\tau_j \in T_i$. If a task $\tau_i \in \mathcal{T}$ is used by multiple functions, for concise expression, multiple copies of the task are defined in the task set $\mathcal{T}$. For example, if task τ_k is used by function F_i and F_j, two tasks (with the same operations of τ_k) are defined in the task set $\mathcal{T}$ to represent τ_k used in function F_i and F_j, respectively.

The relation $\prec$ indicates the dependency relation between two tasks. That is, $\tau_k, \tau_j \in T_i, \tau_k \prec \tau_j$ indicate that task τ_j can start to run only after task τ_k has been completed. The dependency relation is transitive. That is, $\tau_k \prec \tau_j, \tau_j \prec \tau_l \implies \tau_k \prec \tau_l$.

Definition (start task). *A start task of* $(T_i, \prec)$ *is such a task* $\tau_j \in T_i$ *that starts earliest of all the tasks in* T_i*, that is,* $\forall \tau_k \in T_i, j \neq k \implies \tau_j \prec \tau_k$.

Definition (end task). *A end task of* $(T_i, \prec)$ *is such a task* $\tau_j \in T_i$ *that starts latest of all the tasks in* T_i*, that is,* $\forall \tau_k \in T_i, j \neq k \implies \tau_k \prec \tau_j$.

TABLE I. Symbols and definitions

Symbol	Definition
t	system time instant
δ	network precision
$\mathcal{F}$	set of functions of a real-time system
$F_i \in \mathcal{F}$	function of a real-time system, i is the index of the function
r_i	triggered time instant of function F_i
d_i	deadline of function F_i
$\mathcal{T}$	set of all the tasks in the real-time systems
$T_i \subseteq \mathcal{T}$	set of tasks corresponding to function F_i
$\tau_j \in \mathcal{T}$	task, j is index of the task
c_j	computation cost of τ_j
m_j	migration cost of τ_j from a processor to another one
τs_i	start task of task poset $(T_i, \prec)$
τe_i	end task of task poset $(T_i, \prec)$
$\mathcal{P}$	set of processors
$p_a \in \mathcal{P}$	processor, where a is the index of the processor
ps_a	speed of processor p_a
$TS_a \subseteq \mathcal{T}$	task set that can be completed by processor p_a
$\mathcal{N} \subseteq \mathcal{P} \times \mathcal{P}$	set of network channels
$n_{a\to b} \in \mathcal{N}$	network channel from processor p_a to p_b
$ns_{a\to b}$	speed of $n_{a\to b}$

Without losing generality, we assume that there is one start task and one end task of $(T_i, \prec)$, and use τs_i and τe_i to indicate the start task and end task of task poset $(T_i, \prec)$, respectively [1]. Each task has two parameters, $\tau_j = (c_j, m_j)$, where j is the index of the task. c_j is the required computation cost, which means the number of time slots (ticks of processor) needed by a unit speed processor to complete task τ_j; and m_j is the required migration cost for task τ_j migrating from a processor to another one. We use the parameter m_j combined with parameters of network (the details will be explained later) to calculated the overheads of migrating tasks.

C. Processor Set

In multiprocessor real-time systems, different processors are used to execute tasks. We use $\mathcal{P} = \{p_1, p_2, \ldots, p_l\}$ to denote the set of processors, where l is the number of processors. Each processor p_a is a 2-tuple, $p_a = (ps_a, TS_a)$, where a is the index of the processor. Parameter ps_a is the speed of the processor. When task τ_i running on processor p_a, the number of time slots needed for processor p_a to complete task τ_i, represented by *task completion* tc_a^i:

$$tc_a^i = \frac{c_i}{ps_a} \tag{1}$$

TS_a is the task set that can be completed by processor p_a. This parameter is for *heterogeneous* systems in which processors have different architectures, and some tasks can only be executed on some specific processors. If $TS_a = \emptyset$, it means processor p_a cannot be used to execute any task in the system.

Processors have independent local clocks, they are synchronized with each other in the time domain through synchronization protocol. The maximum difference between the

[1] To express a function with many start (end) tasks, we can set a virtual task to be a new start (end) task. Such special task contains empty operation and starts before (after) all the original start (end) tasks.

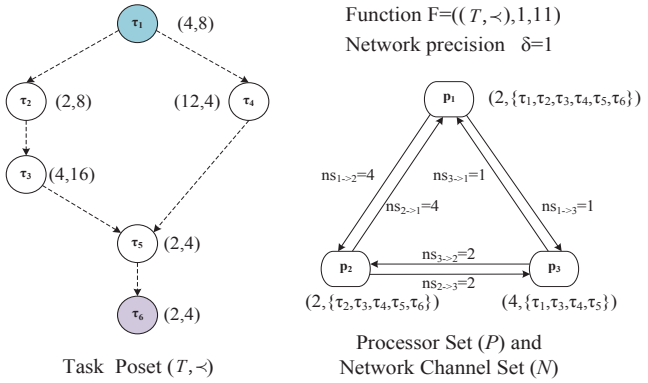

Fig. 1. An example of scheduling for multiprocessor real-time systems

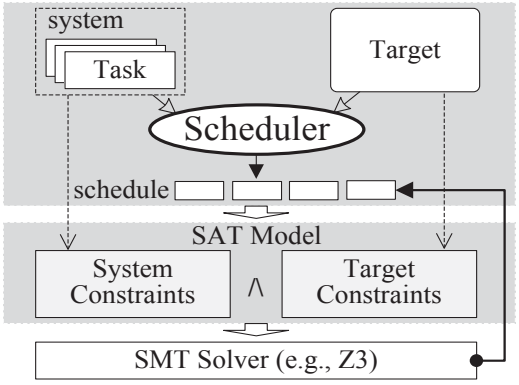

Fig. 2. Overview of the SMT-based scheduling method

local clocks of any two processors in the networked systems is called *network precision* (also called synchronization jitter) which is a global constant. We denote the network precision with δ.

D. Network Channel Set

In multiprocessor real-time systems, processors are connected through network channels. We use $\mathcal{N} \subseteq \mathcal{P} \times \mathcal{P}$ to denote the set of network channels. Symbol $n_{a\to n} \in \mathcal{N}$ denotes the network channel from processor p_a to p_b, where $p_a, p_b \in \mathcal{P}, a \neq b$. Since we consider bi-directional network channel, we have $\forall n_{a\to b} \in \mathcal{N} \implies n_{b\to a} \in \mathcal{N}$. We use $ns_{a\to b}$ to represent the speed of $n_{a\to b}$.

When the data of the computed results of task τ_i migrates from processor p_a to p_b[2], the time slots spent on network channel, represented by $tm^i_{a\to b}$, can be calculated as:

$$tm^i_{a\to b} = \frac{m_i}{ns_{a\to b}} \tag{2}$$

Based on $tm^i_{a\to b}$, we can get the time instant that processor p_b receives the data of task τ_i migrating from processor p_a through network channel $n_{a\to b}$, represented by $r^i_{a\to b}$, as

$$r^i_{a\to b} = s^i_{a\to b} + tm^i_{a\to b} + \delta \tag{3}$$

where, $s^i_{a\to b}$ is the start time of τ_i migrating through network channel $n_{a\to b}$, and δ is the network precision.

E. Assumptions & Example

Applied to this system model, we require that all the parameters of the functions and tasks are known a prior. This requirement makes the model become a generalization of the widely studied *period task model*, in which all the tasks in the system are released periodically. This means our method applies more broadly than other methods which are specified on period task model. To guarantee a certain level of determinacy, in this paper, task preemption is not allowed.

To illustrate the defined system model, an example of scheduling for multiprocessor real-time systems is shown in Fig. 1. In this example, there are three processors p_1, p_2, p_3 in the system. These processors are connected with each other through six network channels $n_{1\to2}, n_{2\to1}, n_{1\to3}, n_{3\to1}, n_{2\to3}, n_{3\to2}$. The network precision δ is 1. In the system, a function $F = ((T, \prec), 1, 11)$ is waiting to be executed on the processors. The task poset of the function is $(T, \prec)$ which consists of six tasks. Task dependency relations are described in a directed acyclic graph. An edge starting from task τ_i to task τ_j represented by a dotted line denotes a dependency relation $\tau_i \prec \tau_j$.

To design scheduling for this kind of system, the schedule synthesis is known to be an NP complete problem. In order to solve such a problem, we propose a scheduling method based on *satisfiability modulo theories (SMT)*. The details are described in the next section.

III. SMT-based Scheduling

A. Satisfiability Modulo Theories

Satisfiability modulo theories (SMT) checks the satisfiability of logic formulas in first-order formulation with regard to certain background theories like linear integer arithmetic or bit-vectors [2]. A first-order logic formula uses variables as well as quantifiers, functional and predicate symbols, and logic operators [3]. A formula F is *satisfiable*, if there is an interpretation that makes F `true`. For example, formula $\exists a, b \in \mathbb{R}, (b > a + 1.0) \wedge (b < a + 1.1)$, where $\mathbb{R}$ is real number set, is satisfiable, as there is an interpretation, $a \mapsto -1.05, b \mapsto 0$, that makes F `true`. On the contrast, a formula F is *unsatisfiable*, if there does not exist an interpretation that makes F `true`. For example, if we define $\exists a, b \in \mathbb{Z}$, where $\mathbb{Z}$ is integer set, the formula $(b > a+1.0) \wedge (b < a+1.1)$ will be unsatisfiable.

For a satisfiability problem that has been formalized by first-order logic formulas, a SMT solver (e.g., Z3, Yices) can be employed to solve such a problem. If all the logic formulas are satisfiable, SMT solver returns the result `sat` and a *solution model* which is an interpretation for all the variables defined in the formulas that makes the formulas `true`. For the case $\exists a, b \in \mathbb{R}$, the model is: $a \mapsto -1.05, b \mapsto 0$. If there is an unsatisfiable logic formula, SMT solver returns the result `unsat` with an empty model, for the case $\exists a, b \in \mathbb{Z}$.

[2]For conciseness, we say "task τ_i migrates from processor p_a to p_b" to mean "the data of the computed results of task τ_i migrates from processor p_a to p_b" in the reset of the paper.

B. Overview of the SMT-based Scheduling

The overview of the SMT-based scheduling is illustrated in Fig. 2. In a real-time system, a schedule (execution order of tasks) is generated by a scheduler. The problem of scheduling can be treated as a satisfiability problem.

In order to use SMT to solve this satisfiability problem, the key work is to formalize the problem using first-order language. We use a *sat model* to represent the formalized problem. This sat model is the set of first-order logic formulas (within linear arithmetic in the formulas) which expresses all the constraints that the desired schedule should satisfy. There are two kinds of constraints: *system constraints* and *target constraints*. System constraints are based on the specific system. For example, if two tasks run on a processor, a schedule should make sure that the execution of these two tasks cannot have overlap in time domain. Target constraint is based on the scheduling target. Specific to this paper, the desired schedule should make all the functions meet their deadlines (completed before deadlines).

After the sat model is constructed, it can be inputted into a SMT solver (e.g., Z3). A *solution model* will be returned by the SMT solver. This solution model gives an interpretation for all the variables defined in the sat model, and under the interpretation, all the logic formulas in the sat model are evaluated as `true`. It means the satisfiability problem represented by the sat model is solved, and based on this interpretation, the desired schedule can be generated.

C. Scheduling Constraints

This subsection describes all the constraints expressed in the sat model.

System Constraints

1) Constraint on start execution time of functions: Task set T_i corresponding to function F_i can start to run only after the function is triggered. That is, the start execution time of the start task of the poset $(T_i, \prec)$ should be larger than the triggered time of function F_i.

$$\forall F_i \in \mathcal{F}, \forall p_a \in \mathcal{P}$$
$$s_a^{\tau s_i} \geq r_i$$

where symbol $s_a^{\tau s_i}$ denotes the start execution time of task τs_i on processor p_a.

2) Constraint on start time of task migration: If a task τ_i migrates from processor p_a to processor p_b through network channel $n_{a\to b}$, it means *i):* task τ_i has been completed by processor p_a; *ii):* τ_i has migrated to processor p_a from another processor. For the first case, task τ_i can start to migrate after it has been completed, and for the second case, task τ_i can start to migrate after it has already migrated to processor p_a.

$$\forall \tau_i \in \mathcal{T}, \forall n_{a\to b} \in \mathcal{N}, \exists n_{c\to a} \in \mathcal{N}$$
$$(s_{a\to b}^i \geq s_a^i + tc_a^i) \vee (s_{a\to b}^i \geq r_{c\to a}^i)$$

where symbol $s_{a\to b}^i$ denotes the start time of task τ_i migrating through network channel $n_{a\to b}$, s_a^i denotes the start execution time of task τ_i on processor p_a.

Algorithm 1 Schedule Synthesis

Input: function set $\mathcal{F}$, task set $\mathcal{T}$, processor set $\mathcal{P}$, network set $\mathcal{N}$
Output: schedule $\mathcal{S}$
1: $\mathcal{A} :=$ `Assert`$(\mathcal{F}, \mathcal{T}, \mathcal{P}, \mathcal{N})$
2: $\mathcal{M} :=$ `CallSMTSolver`$(\mathcal{A})$
3: **if** $\mathcal{M} = \emptyset$ **then**
4: **return** `UNFEASIBLE`
5: **end if**
6: **return** $\mathcal{S} :=$ `GenSch`$(\mathcal{M})$

Algorithm 2 `GenSch()`

Input: solution model $\mathcal{M}$
Output: schedule $\mathcal{S}$
1: $\mathcal{S} := \emptyset$
2: **for all** $F_i \in \mathcal{F}$ **do**
3: sort task set T_i, such that $\langle \tau_1, \tau_2, \ldots, \tau_n \rangle$ is a permutation with $\tau_{j_1} \not\prec \tau_{j_2}$ for $1 \leq j_1 < j_2 \leq n$, where n is the number of tasks in T_i
4: determine s_a^1 based on *constraint 8*, $\mathcal{S} := \mathcal{S} \cup \{s_a^1\}$
5: **for all** $\tau_j \in T_i \backslash \tau_1$ **do**
6: determine s_a^j based on *constraints 2 and 3*
$\mathcal{S} := \mathcal{S} \cup \{s_a^j\}$
7: **if** τ_j has migration among processors **then**
8: determine $s_{b\to c}^j$ based on *constraints 2 and 3*
$\mathcal{S} := \mathcal{S} \cup \{s_{b\to c}^j\}$
9: **end if**
10: **end for**
11: **end for**
12: **return** $\mathcal{S}$

3) Constraint on task dependency: For processor p_a, if $\tau_i \prec \tau_j$, task τ_j can start to run only after τ_i has been completed. Similar to the constraints on start time of task migration, there are two cases. *i):* task τ_i has been completed by processor p_a; *ii):* task τ_i has migrated to processor p_a from another processor. For the first case, τ_j can start to run after τ_i has been completed, and for the second case, τ_j can start to run after τ_i has already migrated to processor p_a.

$$\forall \tau_i, \tau_j \in \mathcal{T}, \forall p_a \in \mathcal{P}, \exists n_{b\to a} \in \mathcal{N}$$
$$\tau_i \prec \tau_j \implies (s_a^j \geq s_a^i + tc_a^i) \vee (s_a^j \geq r_{b\to a}^i)$$

4) Constraint on critical resources: In multi-tasks system, some tasks may require a same critical resource. A reasonable scheduling sequence should guarantee that no multiple tasks access to a critical resource at the same time. We use $\tau_i \sim \tau_j$ to represent task τ_i and τ_j require a same critical resource when they are running.

$$\forall \tau_i, \tau_j \in \mathcal{T}, i \neq j, \forall p_a, p_b \in \mathcal{P}$$
$$\tau_i \sim \tau_j \implies (s_a^i \geq s_b^j + tc_b^j) \vee (s_a^j \geq s_b^i + tc_b^i)$$

5) Constraint on processors: A processor can execute only one task at a time. This is interpreted as: there is no overlap of the execution time of any two tasks.

$$\forall \tau_i, \tau_j \in \mathcal{T}, i \neq j, \forall p_a \in \mathcal{P}$$
$$(s_a^i \geq s_a^j + tc_a^j) \vee (s_a^j \geq s_a^i + tc_a^i)$$

6) Constraint on network channels: A network channel can transfer data of only one task at a time. That is, there is no overlap of the migration time of any two tasks on a network channel.

$$\forall \tau_i, \tau_j \in \mathcal{T}, i \neq j, \forall n_{a\to b} \in \mathcal{N}$$
$$(s_{a\to b}^i \geq s_{a\to b}^j + tm_{a\to b}^j) \vee (s_{a\to b}^j \geq s_{a\to b}^i + tm_{a\to b}^i)$$

7) Constraint on heterogeneous processors: In heterogeneous systems, processors have different architectures, some tasks can only be executed on some specific processors. For tasks that cannot be executed on some processors, the start execution time of the tasks in such processors are set to $+\infty$, which means the tasks will never start to run on these specific processors.

$$\forall p_a \in \mathcal{P}, \forall \tau_i \in \mathcal{T} - TS_a$$
$$s_a^i = +\infty$$

Target Constraints

8) Constraint on deadlines of functions: For every triggered function, the desired schedule should make sure that the function can be completed before its deadline.

$$\forall F_i \in \mathcal{F}, \exists p_a \in \mathcal{P}$$
$$s_a^{\tau e_i} + tc_a^{\tau e_i} \leq d_i$$

where symbol $s_a^{\tau e_i}$ is the start execution time of task τe_i on processor p_a, and $tc_a^{\tau e_i}$ is the number of time slots needed for processor p_a to complete task τe_i.

D. Schedule Synthesis

After all the constraints are defined, we now can employ a SMT solver to generate the desired schedule. The process of schedule synthesis is summarized in Alg. 1. Function $\mathcal{A} := \texttt{Assert}(\mathcal{F}, \mathcal{T}, \mathcal{P}, \mathcal{N})$ (line 1) interprets the constraints defined in section III-C as *assertions* (boolean formulas that can be inputted into a SMT solver) based on the system model. The variables of these boolean formulas are the start time of task execution on processor, s_a^j, and start time of task migration through network, $s_{b\to c}^j$, for $\forall F_i \in \mathcal{F}, \forall \tau_j \in T_i, \forall p_a \in \mathcal{P}, \forall n_{b\to c} \in \mathcal{N}$. Function $\texttt{CallSMTSolver}(\mathcal{A})$ (line 2) calls a SMT solver to find a solution model $\mathcal{M}$ for $\mathcal{A}$, which contains the interpretation for all the variables s_a^j and $s_{b\to c}^j$ defined in the set of assertions $\mathcal{A}$. If the solution model does not exist ($\mathcal{M} = \emptyset$) (line 3), message `UNFEASIBLE` will be returned (line 4), which means there does not exist a schedule can make all the functions meet their deadlines. Otherwise, if the solution model exists ($\mathcal{M} \neq \emptyset$), based on $\mathcal{M}$, function $\texttt{GenSch}(\mathcal{M})$ generates the desired schedule $\mathcal{S}$ which can make all the functions meet their deadlines (line 6).

Alg. 2 describes the details of function $\texttt{GenSch}()$. It returns the schedule $\mathcal{S}$ which is a set of variables selected from the solution model $\mathcal{M}$. $\mathcal{M}$ contains the interpretation for all the variables s_a^j and $s_{b\to c}^j$ defined in the set of assertions $\mathcal{A}$. As the exist of existential quantifier $\exists$, in constraints 2, 3, and 8, we need to further select s_a^j and $s_{b\to c}^j$ that can form the desired schedule. For example, in constraint 8, we should find which processor (i.e., to determine the value of processor index a) can make the logical formula $\forall F_i \in \mathcal{F}, s_a^{\tau e_i} + tc_a^{\tau e_i} \leq d_i$ be evaluated as `true`. After these judgments, we can determine s_a^j and $s_{b\to c}^j$ that can form the desired schedule. For each function F_i, $\texttt{GenSch}()$ first sorts its corresponding task set T_i, such that $\langle \tau_1, \tau_2, \ldots, \tau_n \rangle$ is a permutation with $\tau_{j_1} \not\prec \tau_{j_2}$ for $1 \leq j_1 < j_2 \leq n$ (line 3). Follow this sorted order, the first task τ_1 is the end task τe_i of function F_i. Based on constraint 8, the value of processor index a can be determined. That is, we can determine s_a^1 (i.e., $s_a^{\tau e_i}$) (line 4). For other tasks in T_i, as $\tau_{j_1} \not\prec \tau_{j_2}$ for $1 \leq j_1 < j_2 \leq n$, after the value of s_a^1 is determined, based on constraints 2 and 3, the values of s_a^j and $s_{b\to c}^j$ (only for tasks which migrates among processors) can be determined (line 5-10). After this, the desired schedule $\mathcal{S}$ can be generated (line 12).

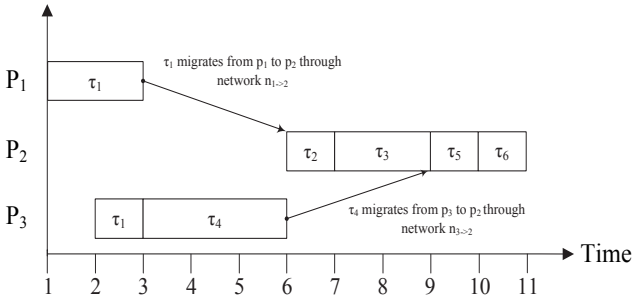

Fig. 3. The scheduling result for example shown in Fig. 1 by using the proposed SMT-based scheduling

E. Scheduling Results

Recall the example shown in Fig. 1. Based on the schedule synthesis described in Alg. 1, we can get the solution model $\mathcal{M}$ which defines the values of the start time of task execution on processor, s_a^j, and the start time of task migration through network, $s_{b\to c}^j$, for $\forall F_i \in \mathcal{F}, \forall \tau_j \in T_i, \forall p_a \in \mathcal{P}, \forall n_{b\to c} \in \mathcal{N}$. From the model $\mathcal{M}$, through function $\texttt{GenSch}()$ as shown in Alg. 2, we can get the scheduling result as shown in Fig. 3. This scheduling sequence can make the function meet its deadline. Some characteristics of this scheduling sequence should be noticed:

- Task τ_1 has been executed on processor p_1 from system time $t = 1$ to $t = 3$, and it has also been executed on processor p_3 from system time $t = 2$ to $t = 3$. This means, the SMT-based scheduling method can handle the parallel execution of tasks, and can make a task repeatedly run on different processors when such repeated execution is necessary.
- Task τ_2 runs on processor p_2 from $t = 6$ to $t = 7$. Although task τ_2 needs the computed results obtained from completing task τ_1, such computed results not only can be obtained by completing task τ_1 on processor p_2 itself, but also can be obtained by transferring the computed results from other processor that has completed task τ_1. Specified to this example, at system time $t = 6$, processor p_2 gets the computed results of task τ_1 from processor p_1.

IV. Simulation & Evaluation

In this section, we present the results of simulations which are conducted to study the performance of the SMT-based scheduling method. We have implemented a prototype tool for the proposed SMT-based scheduling based on the system model, constraints formulation, and schedule synthesis described above. The underlying SMT solver employed by the tool is Z3 [6], which is a state-of-the art SMT solver.

A. Simulation Settings

The metric used to evaluate the scheduling performance is: *simulation runtime* which denotes the time of the SMT-based

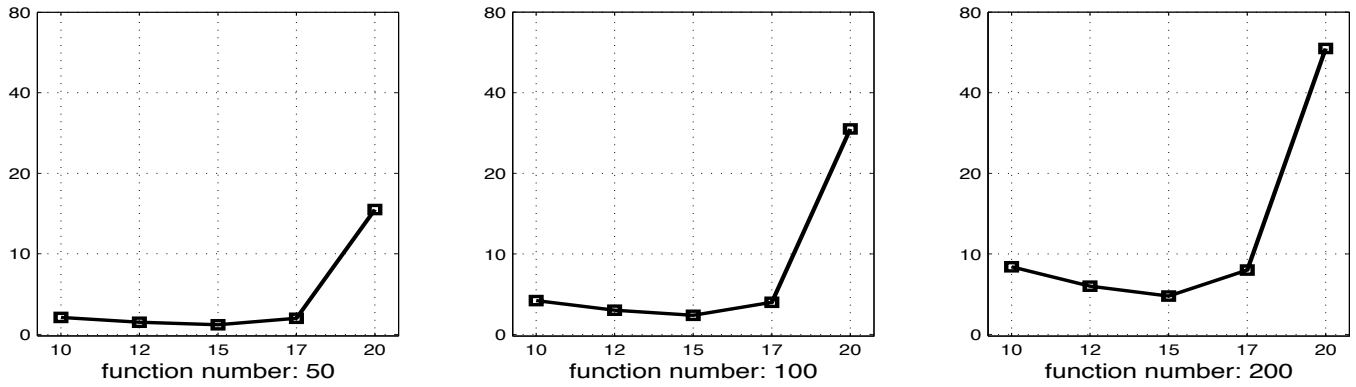

Fig. 4. Simulation runtime (y-axis, in second) of the SMT-based scheduling for systems with 3 processors (The x-axis is the values of λ.)

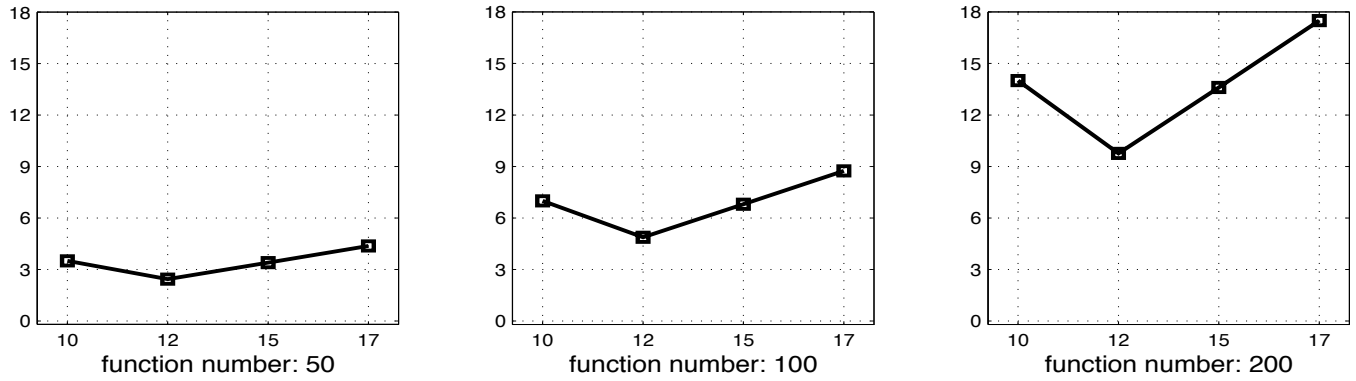

Fig. 5. Simulation runtime (y-axis, in second) of the SMT-based scheduling for systems with 4 processors (The x-axis is the values of λ.)

scheduling method scheduling all the input tasks. The input functions are generated according to uniform distribution with arriving rate λ which represents the number of tasks that arrive in the system per 100 time units. Note that, parameter λ can be used to adjust system workload. The numbers of tasks within a function are randomly generated, and the average value approximates to 6. The dependency relation of the tasks is randomly assigned. For each task τ_j, c_j and m_j vary in $[1\ 18]$. The assignments of deadlines of functions d_i are according to the equation: $d_i = r_i + \lfloor sf_i * C_i \rfloor$, where C_i is the total computation cost of all the tasks in the function, and sf_i is the slack factor that indicates the tightness of task deadline, for each function, sf_i varies in $[0.3\ 3]$. For the network channels, we consider a mesh topology (all the processors are connected with each other through network channels) in the simulation, and the network speed varies in $[1\ 6]$. There are many parameters that can affect the simulation runtime. We mainly study the impacts of the number of functions, number of processors, and the parameter λ. The time-out for each experiment is set to 10 minutes after which the scheduling problem is deemed unfeasible. All the simulations are run on a 64bit 4-core 2.5 GHz Intel Xeon E3 PC with 32GB memory.

B. Evaluation

The simulation results are shown in Fig. 4 and Fig. 5. The values shown in the figures are the average value of running simulation 100 times. Fig. 4 shows the results for systems with three processors. For 50 functions, when λ is in interval $[10\ 17]$, the simulation runtimes are from 2 to 4 seconds, and the difference of the runtimes is not obvious. However, when λ increases to 20, the simulation runtime increases to 17 seconds, which is quite a big increase. Moreover, we also conduct simulations for λ with 25, and the simulation is time-out (over 10 min). This is because, with different λ, the system workload condition is different. With bigger value of λ, more functions (more tasks) are waiting to be executed on processors at a time. As described in section III-C, the scheduling constraints *Constraint on processors* and *Constraint on network channels*, need to guarantee that no overlap of the execution time of any two tasks on a processor and no overlap of the migration time of any two tasks on a network channel. The increasing number of waiting tasks will need much more calculation for the SMT solver.

For example, when λ is 10, the average number of tasks waiting to be executed at a time is 6. For these 6 tasks, SMT will conduct calculation to find a solution model. As the total input number of tasks is around 300 (total number of input functions is 50), such SMT calculation will be conducted about 50 (300/6) times. When λ is 20, the the average number of tasks waiting to be executed at a time is 100, although the times for SMT to conduction calculation is only 6, but for 100 tasks, the calculation for SMT to find a solution model is much more complicated, which results in a long time calculation.

Fig. 5 shows the results for systems with four processors. Compared with Fig. 4, the results for $\lambda = 20$ are not shown in the Fig. 5, as when $\lambda = 20$, the simulation is time-out.

From this we can know that the number of processors can also obviously affect the SMT calculation in addition to λ. For the total input function number, with increasing of its value, the simulation runtime increases in proportion. For example, in Fig. 5, with a certain λ, the simulation runtime for 100 total input functions is twice as the time for 50 total input functions. This is consistent with our previous analysis that the total input number of functions decides the times of the SMT calculations to find a solution model for all the input functions.

V. Concluding Remarks

In this paper, to solve the scheduling problem for multiprocessor real-time systems, a SMT-based scheduling method is proposed. In the method, the problem of scheduling is treated as a satisfiability problem. After using first-order language to formalize the satisfiability problem, a SMT solver is employed to solve such a problem. An optimal schedule can be generated based on a solution model returned by the SMT solver. The correctness of this method and the optimality of its generated schedule are straightforward. Moreover, in the SMT-based scheduling method, we define the scheduling constraints as system constraints and target constraints. Such design makes the proposed method apply more widely compared with existing methods.

For the future work, in order to study the performance of the SMT-based scheduling method in a real application, we would like to implement the proposed method in a real multiprocessor real-time system.

References

[1] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with EDF scheduling," *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1250–1258, Apr. 2009.

[2] C. Barrett, R. Sebastiani, R. Seshia, and C. Tinelli, "Satisfiability modulo theories," Handbook of Satisfiability, vol. 185. IOS Press, 2009.

[3] L.d. Moura N. Bjrner, "Satisfiability Modulo Theories: An Appetizer," *Formal Methods: Foundations and Applications*, vol. 5902, pp. 23–26, 2009.

[4] S.S. Craciunas and R.S. Oliver, "SMT-based Task- and Network-level Static Schedule Generation for Time-Triggered Networked Systems," *Proc. 22th Int. Conf. on Real-Time Networks and Systems*, NY, USA, pp. 45–54, October, 2014.

[5] R.I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 5, pp. 35:1–35:44, Oct. 2011.

[6] L. Moura and N. Bjrner, "Z3: an efficient SMT solver," *Proc. 14th Int. Conf. on Tools and Algorithms for the Construction and Anal. of Syst.*, Budapest, Hungary, LNCS 4963, pp. 337–340, Springer-Verlag, 2008.

[7] B. Dutertre, "Yices 2.2," *Proc. 26th Int. Conf. on Comput. Aided Verification*, Vienna, Austria, LNCS 8559, pp. 737–744, Springer International Publishing, 2014.

[8] S.K. Baruah, N. Cohen, G. Plaxton, and D. Varvel, "A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.

[9] J. Anderson and A. Srinivasan, "Early-release fair scheduling," *Proc. of the Euromicro Conference on Real-Time Systems*, 2000.

[10] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty, "Task- and network-level schedule co-synthesis of Ethernet-based time-triggered systems," *Proc. of ASP-DAC*, 2014.

[11] T. Pop, P. Eles, and Z. Peng, "Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems," *Proc. of CODES*, 2002.